gotcha

A **gotcha** is an *unexpected side effect, behavior, consequence, requirement, etc.*

See Also

Dangerous constructs

Description

As with any language, the syntax and semantics of Tcl can catch a programmer off-guard. This page is describes patterns and behaviours that can be considered as **gotcha**s, and is somewhat arranged by prominence and severity.

Gotchas

The Octal Bug

See Tcl and octal numbers

set and namespaces

Twylite 2012-12-12 PYK 2019-12-09: set might set a global variable rather than a variable in the current namespace. See Dangers of Creative Writing.

```
set foo 10
namespace eval bar { set foo 20 ; set bar 30 }
puts $foo ;# puts: 20
puts $bar ;# error: can't read "bar": no such variable
```

Unbraced Expressions

The following script will never end:

```
while [llength $args] {
    lassign args arg
}
```

It should instead be:

```
while {[llength $args]} {
    lassign args arg
}
```

See while and also Brace Your Expressions.

Assuming that a value is a dictionary

In the following example the fact that $dict is not a dictionary does not trigger an error:

```
set dict hello
dict exists $dict Bob ;# -> 0
```

Assuming that a value is a list

Another Tcl **gotcha** is to hand arbitrary strings, read from the user or a file/command, directly into a list operation without first ensuring that the contents is, in actuality, a list.

DKF: The best way to deal with such user input, where users aren't expecting to write a Tcl list in the first place, is to use a sanitizing command to convert the input into a proper list. Examples of sanitizing commands can include split, splitx and (my favorite) this:

```
set elements [regexp -all -inline {\S+} $line]
```

Assuming that a variable is an array

Using array unset on a variable that isn't array does nothing, and does not trigger an error:

```
set array 5
array unset array
set array ;# -> 5
```

lsearch default matching style is -glob

Forgetting to use -exact when that's what is intended can lead to surprises.

switch and its Arguments

RS One possible gotcha is switch -- always use -- before the switch variable, since if the value of the switch variable starts with -, you'll get a syntax error.

AMG: This isn't required by Tcl 8.5 onward.

Delimiting Options from Arguments

KPV Also, comments within switch, while possible, are tricky.

RS 2010-05-10: A similar gotcha is in the **text search** subcommand - although the misunderstanding could be avoided by counting non-optional arguments from the end,

```
set whatever -this
$t search $whatever 1.0
```

raises an error that -this is an undefined switch. For robustness, use

```
$t search -- $whatever 1.0
```

if the slightest possibility exists that $whatever might start with a dash.

PYK 2017-05-19: scripted lists makes comments within a switch possible, but it needs to move into the Tcl implementation level to be performant.

Interpretations by expr

RS 2010-02-24: Yet another gotcha we ran into last night: Consider a function

```
proc f x {
    if {$x == 00000000} {
        puts "$x is NULL"
    }
}
```

which reported:

```
0E123456 is NULL
```

How so? Bug? No -- feature. With the == comparison operator, the operands are tried to match as integers, floats, or then as strings. And $x in this case, though meant as a pointer in hex, could be parsed as float - with the numeric value of 0, which is numerically equivalent to 00000000. The solution of course was to use the eq operator instead.

AMG: Another issue with expr interpreting stuff happens when said stuff was already interpreted by Tcl. This creates performance, security, and correctness problems. Always brace your expr-essions!

Another example:

```
expr {{} < 0} ;# -> 1
expr {{} > 0} ;# -> 0
expr {{} == 0} ;# -> 0
```

For the < and > operators, {} is interpreted as an integer, 0. For the == operator, 0 is interpreted as a string.

string is and the empty string

string is always returns 1 for the empty string. To avoid this, use -strict

Confirming that a value is an integer decimal notation

The string is commands accept all the numeric notations that expr does (see The Octabug, so they aren't up to the job by themselves. Here is one way to do it:

```
set mynumber 0x11
expr {[string is entier -strict $mynumber] && [scan $mynumber %d mynumber] > 0}
```

Magic characters in filenames and arguments

The open command has special handling when its filename argument begins with |. The exec command has special handling when any argument begins with | or < or > or 2> or when the final argument is &. Normally this is fine, but if one of these arguments is not under the programmer's control, unexpected behavior can result.

One may work around open by adjusting the filename if it begins with |.

```
set tmpf $filename
if {[string match |* $tmpf]} {set tmpf [file join . $tmpf]}
set channel [open $tmpf]
```

There is a proposal (TIP 424) for changes to exec, but as of this writing (May 2020), it has not been implemented in the core language.

Partial sub-command resolution in namespace ensembles

```
namespace eval table {
    namespace export *
    namespace ensemble create
}

proc {table::spoon fork} x {
    return $x
}
```

Even though there is no spoon, there is:

```
% puts [table spoon huh?]
huh?
```

To change this behavior:

```
namespace ensemble configure table -prefixes no
```

Nested vwait

See Avoiding Conflicting vwait's on the vwait page

Starving Queues with after

See after x after idle on the after page

chan copy bytes or characters

If the encoding of the two channels matches, the given size is in bytes, otherwise it is in characters.

Misc

Larry Smith: I was expecting a new control construct of some sort.

```
gotcha {
  ...
} now {...}
```

Updated 2022-01-21 22:03:09